John von Neumann Institute for Computing


# Hash Based Adaptive Parallel Multilevel Methods with Space-Filling Curves

Michael Griebel, Gerhard Zumbusch

published in

*NIC Symposium 2001, Proceedings*,
Horst Rollnik, Dietrich Wolf (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **9**, ISBN 3-00-009055-X, pp. 479-492, 2002.



http://www.fz-juelich.de/nic-series/volume9

# Hash Based Adaptive Parallel Multilevel Methods with Space-Filling Curves

**Michael Griebel**[1] **and Gerhard Zumbusch**[1]

Institut für Angewandte Mathematik, Universität Bonn
Wegelerstr. 6, 53115 Bonn, Germany
*E-mail: {griebel, zumbusch}@iam.uni-bonn.de*

The solution of partial differential equations on a parallel computer usually follows the data parallel paradigm. The grid is partitioned and mapped onto the processors. In this paper a parallelisable and cheap method based on space-filling curves is proposed. The partitioning is embedded into the parallel solution algorithm using multilevel iterative solvers and adaptive grid refinement. Numerical experiments on two massively parallel computers prove the efficiency of this approach.

## 1 Introduction

In this paper, we consider parallel versions of adaptive multigrid solvers and adaptive sparse grid discretisations for elliptic partial differential equations. The multigrid method operates on a finite difference discretisation on quad-tree and oct-tree meshes, which are obtained by adaptive mesh refinement. The adaptive sparse grid discretisation is also based on a finite difference scheme and is well suited for higher dimensional problems. A fast parallel load balancing strategy for both approaches is proposed which is defined by a space-filling Hilbert curve. It is furthermore applicable to arbitrary shaped domains. Some numerical experiments demonstrate the parallel efficiency and scalability of the approach.

## 2 Multigrid Solver

Our goal is to solve an elliptic partial differential equation efficiently. The PDE is discretised by finite differences on a 1-irregular quad-tree or oct-tree grid. Here, we set up the operator as a set of difference stencils from one node to its neighboring nodes in the grid, which can be easily determined: Given a node, its neighbors can be only on a limited number of levels. The distance to the neighbor is determined by the refinement level the nodes share, see Figure 1.

So pure geometric information is sufficient to apply the finite difference operator to some vector. Hence we avoid the storage of the stiffness matrix or any related information. For the iterative solution of the equation system, we have to implement matrix multiplication, which is to apply the operator to a given vector.

We use an additive version of the multigrid method for the solution of the equation system, i.e. the so called BPX preconditioner[1]. This requires an outer Krylov iterative solver. The BPX preconditioner has the advantage of an optimal $\mathcal{O}(1)$ condition number and an implementation of order $\mathcal{O}(n)$, which is optimal, even in the presence of degenerate grids. Furthermore, this additive version of multigrid is easier to parallelise than multiplicative multigrid versions.

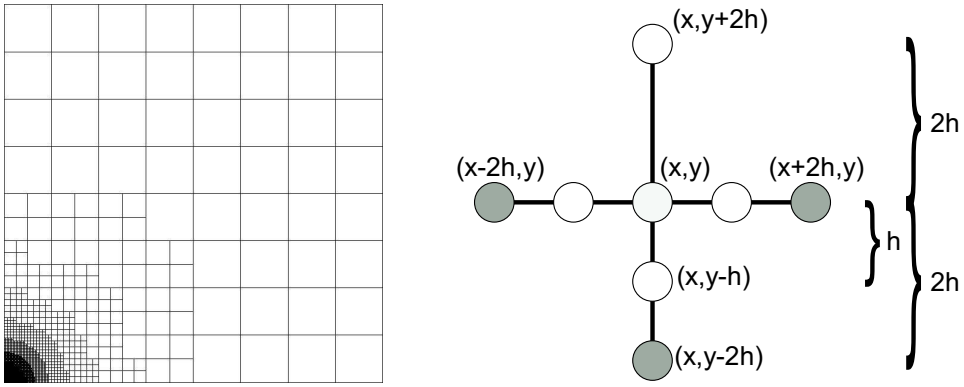


Figure 1. Adaptively refined quad-tree grid (left) and a finite difference stencil (right)

The straightforward implementation is analogous to the implementation of a multigrid V-cycle. However, the implementation with optimal order is similar to the hierarchical basis transformation and requires one auxiliary vector[2]. Two loops over all nodes are necessary, one for the restriction operation and one for the prolongation operation. They can be both implemented as a tree traversal or alternatively by iterating over the nodes in the right order.

A parallel version of multiplicative multigrid usually is based on a partition of all nested grids. The domain $\Omega$ is decomposed into several sub-domains $\Omega_j$, which induces partitions of all grids. Each processor holds a fraction of each grid in such a way that these fractions of each grid form a nested sequence. Hence each operation on a specific level is partitioned and mapped to all processors. Furthermore the communication during grid transfer operations is small because of nested sequences on a processor. This means that one has to treat global problems on each level, which are partitioned to all processors. The intra-grid communication has to be small, i.e. the number of nodes on the boundary of the partition should be small. Furthermore the amount of work on coarse grid levels usually is small and each processor does not compute much. There are several strategies to deal with the coarse grid problem in general, such as to centralize the computation on a master processor, to perform identical computations on all processors or to modify the coarse grid correction step.

A static partition of the domain into strips or squares can be used for uniform grids and has been used for the first parallel multigrid implementations[3,4], see also the survey Ref. 5. In contrast to the geometry oriented parallelisation of multiplicative multigrid methods, the additive multigrid version or additive multilevel preconditioners can be parallelised in a more flexible way. The overall workload has to be partitioned, but we do not have to consider individual levels. Here, the communication takes place in a single step for all nodes, which are located on the boundary of at least one grid of the nested sequence. The multilevel BPX preconditioner for a uniform grid has been parallelised[6,7]. These approaches can easily be generalized to block-structured grids.

## 3 Sparse Grids

Before we consider the parallelisation in more detail, we introduce our second discretisation strategy, which is especially well suited for higher dimensional problems. Sparse grids are a multi-dimensional approximation scheme, which is known under several names such as 'hyperbolic crosspoints', 'splitting extrapolation' or as a boolean sum of grids. Probably Smolyak[8] was the historically first reference. Directly related to the boolean construction of the grids was the construction of a multi-dimensional quadrature formula. Both, quadrature formulae and the approximation properties of such tensor product spaces were subject to further research, see Temlyakov[9] and others[10–13]. The curse of dimension was also subject to general research on the theoretical complexity of higher-dimensional problems. For such reasons, sparse grids play an important role for higher-dimensional problems. Besides the application to quadrature problems, sparse grids are now also used for the solution of PDEs. They were introduced for the solution of elliptic partial differential equations by Zenger[14], where a Galerkin method, adaptive sparse grids and tree data structures were discussed. At the same time a different discretisation scheme based on the extrapolation of solutions on several related, regular grids was proposed, the 'combination' technique[15].

In this paper we consider a finite difference (FD) scheme[16,17]. It is simpler to implement and to apply to different types of equations, but there is not that much known analytically yet. We focus on the parallelisation of such a FD scheme on sparse grids. So far only the 'combination' technique can be run on a distributed-memory parallel computer[18], which is essential for many large scale simulations. The advantage of the FD scheme is the flexibility of the grid. The sparse grid may be refined adaptively, while the 'combination' technique relies on regular sparse grids. However, the adaptive grid refinement poses a severe load-balancing problem, which has to be resolved during runtime of the simulation.

The multi-dimensional approximation scheme of sparse grids can be constructed as a subspace of the tensor-products of one-dimensional spaces represented by a hierarchical multi-resolution scheme, such as the hierarchical basis, see the historical reference Faber[19], or generally any basis system of pre-wavelets or wavelets[20]. Each one-dimensional basis function can be derived from a model function $\phi$ by a scaling of $2^{-l}$ (also called level $l$) and a translation by a multiple of $2^{-l}$. In the case of the piecewise linear hierarchical basis, the model function $\phi$ is the well known hat function. We denote the one-dimensional space of functions up to level $l$ by $T_l$. On level $l$, the standard grid space can be written as

$$\hat{T}_l = \langle T_i \otimes T_j \otimes \ldots \rangle_{i,j,\ldots \leq l} \,. \tag{1}$$

In contrast to the regular grid, the corresponding sparse grid space consists of fewer functions. On level $l$, it can be written as

$$\tilde{T}_l = \langle T_i \otimes T_j \otimes \ldots \rangle_{i+j+\ldots \leq l} \,. \tag{2}$$

This is a subset of the regular grid space. A regular grid has about $2^{d \cdot l}$ nodes, which is substantially more than the $2^l \cdot l^{d-1}$ nodes of the sparse grid.

The major advantage of sparse grids compared to regular grids is their smaller number of nodes (or grid points) for the same level $l$ and resolution $2^{-l}$. This is especially true in higher dimensions $d \gg 1$.

Of course, the question whether sparse grids have an advantage compared to regular grids does also depend on the discretisation accuracy of a solution obtained on a grid.

Furthermore the number-of-operations complexity is of interest, because it is an estimate for the computing time a specific algorithm needs. For details see Ref. 16, 17.

We define the hierarchical transformation $\mathbf{H}$ as the hierarchical basis transformation on the regular grid from nodal values to hierarchical values, which are restricted to the sparse grid nodes. All wavelet-type basis functions provide such fast $\mathcal{O}(n)$ transformation to and from the nodal basis representation. The transformation is especially simple for the one-dimensional hierarchical basis: Given the nodal values $u_j$ with $j = 0, 1, \ldots, 2^{l+1}$, the hierarchical representation for interior points can be obtained by

$$\hat{u}_j \;=\; u_j - \frac{1}{2}\big(u_{\text{left father}} + u_{\text{right father}}\big) \tag{3}$$

and the boundary nodes $u_0$ and $u_{2^{l+1}}$ remain unchanged. The nodal values are replaced by their hierarchical excess or deterioration, compared to the value obtained by interpolation between on the next coarser level grid. The inverse transformation can be implemented similarly. However, the coarse nodes have to be computed before the finer grid nodes. Furthermore, the transformation can be implemented in place, without an auxiliary vector. The hierarchical basis transformation $\mathbf{H}$ is also abbreviated by the stencil $[1/2\ 1\ 1/2]$.

Based on the hierarchical basis transformation $\mathbf{H}$, we define the action of a one-dimensional finite difference operator for the discretisation of a differential operator: We apply the associated standard difference stencil $\mathbf{D}_i$ along the $x_i$-axis to values located on the sparse grid nodes in a specific basis representation. To this end the values are given in nodal basis in direction $i$ and in hierarchical basis representation in all other directions $I \setminus \{i\}$. The associated transformation is denoted by $\mathbf{H}_{I\setminus\{i\}}$. The stencil $\mathbf{D}_i$ for each node itself is chosen as the narrowest finite difference stencil available on the sparse grid. It is equivalent to the corresponding stencil on a regular, anisotropic refined grid. The finite difference stencil can be a 3-point Laplacian $\frac{1}{h^2}[1\ -2\ 1]$, an upwind-stabilized discretisation of the convective term $\frac{\partial}{\partial x_i}$, some variable coefficient operators and so on. In nodal values the finite difference operator reads

$$\frac{\partial^2}{\partial x_i^2} u \;\approx\; \mathbf{H}^{-1}_{I\setminus\{i\}} \circ \mathbf{D}_{ii} \circ \mathbf{H}_{I\setminus\{i\}} u\,. \tag{4}$$

A general difference operator is then obtained by dimensional splitting. A Poisson equation, as a simple example, can be discretised in nodal basis representation as usual as a sum of operators (4). Here the one-dimensional difference operators $\mathbf{D}_i$ may be chosen as a three point centered Laplacian $\frac{1}{(x_{i+1}-x_{i-1})^2}[1\ \ -2\ \ 1]$. On adaptively refined grids, the nearest neighbor nodes are chosen, which may lead to asymmetric stencils, i.e. non-uniform one-dimensional stencils. Further higher order modifications of the stencils have been tested, too. In the presence of a transport term in the equation, the unsymmetry is believed to be no problem. There are ways to create discretisations for all kinds of equations, e.g. for general diffusion problems, convection-diffusion problems, reaction-diffusion problems, for the Navier-Stokes equations[17] or for hyperbolic conservation laws[21]. Known facts on consistency and stability of this scheme are summarised in Ref. 17.

## 4 The Load-Balancing Problem

Finite-Element, Finite-Volume and Finite-Difference methods for the solution of partial differential equations are based on meshes. The solution is represented by degrees of freedoms attached to certain locations on the mesh. Numerical algorithms operate on these degrees of freedom during steps like the assembly of a linear equation system or the solution of an equation system. A natural way of porting algorithms to a parallel computer is the data distribution approach. The mesh with attached degrees of freedom is decomposed into several partitions and mapped to the processors of the parallel computer. Accordingly also the operations on the data are partitioned. Goals of a partitioning scheme are load-balancing and little communication between the processors. Sometimes also singly-connected partitions are required. If the partitions are determined during run-time, furthermore a fast partitioning scheme itself is sought. This is e.g. the case within adaptive mesh refinement of a PDE solver.

The partitioning problem in general is $NP$-hard[22]. There are many heuristics based on graph connectivity or geometric properties to address this problem[23–27]. In practice fast heuristics are known. However, there is not much known about the general quality of these methods[28]. In contrary there exist examples, where single heuristics give really bad results.

In this paper we propose a specific geometry based heuristic with space-filling curves. It is cheap and helps to simplify the implementation of parallel algorithms[29–34]. We are interested in bounds for the quality of the partitions. This will lead us to general estimates on the parallel performance of advanced numerical algorithms on these partitions.

## 5 Space-Filling Curves

First we have to define curves. The term curve shall denote the image of a continuous mapping of the unit interval to the $\mathbb{R}^d$. Mathematically, a curve is space-filling if and only if the image of the mapping does have a classical positive $d$-dimensional measure. The curve fills up the whole domain. For reasons of simplicity we restrict our attention to simple domains. We are interested in a mapping

$$f \ : \ [0,1] =: I \ \mapsto \Omega \subset \mathbb{R}^d, \qquad f \text{ continuous and surjective.} \tag{5}$$

There are classical space-filling curves like the Hilbert-, the Peano- and the Lebesgue-curve[35]. However, we will also construct special space-filling curves on an unstructured mesh.

A space-filling curve can also be used for the inverse mapping $f$ from a domain $\Omega \subset \mathbb{R}^d$ to the unit interval $I$. This means that we can map geometric entities in $\mathbb{R}^d$, such as elements or nodes, to the one dimensional interval. Entities, which are neighbors on the interval, are also neighbors in the volume $\mathbb{R}^d$. Unfortunately the reverse cannot be true and neighbors in the volume may be separated through the mapping.

However, we can solve the resulting one-dimensional partition problem: We cut the interval $I$ into disjoint sub-intervals $I_j$ of equal workload with $\bigcup_j I_j = I$. This gives perfect load-balance and small separators between the partitions. The partition $f(I_j)$ of the domain $\Omega$ induced by the space-filling curve with $\bigcup_j f(I_j) \supset \Omega$ also gives perfect load-balance. However, the separators $\partial f(I_j) \setminus \partial\Omega$ (boundary of the geometrical sets) are larger than the optimal separators in general as we will see in the following.

## 6 Quality of a Partition

We use a basic performance model for a distributed memory computer. The execution time of a program consists of computing time, which is proportional to the number of operations on a processor, and of communication time. Communication between the processors is implemented with message passing through some network and requires time linear in the size of data $t = t_{\text{startup}} + n * t_{\text{bandwidth}}$.

We consider $\mathcal{O}(n)$ algorithms which are linear in the size of data $n$, e.g. FEM matrix assembly for $n$ finite elements, sparse matrix multiply or components of a multigrid algorithm such as a grid transfer or smoother[23,36]. The parallel computing time is $C_1 \cdot n/p$ for a partition of $n$ data onto $p$ processors. We call $v := n/p$ the *volume*. The runtime depends on the communication time. The data to be transferred is proportional to the separator or surface $s_j$ of the partition $s_j := \partial f(I_j) \setminus \partial\Omega$. Altogether we have

$$t = C_1 \frac{n}{p} + C_2(t_{\text{startup}} + s * t_{\text{bandwidth}}) \,. \tag{6}$$

This model suggest that we have to minimize the surface to volume ratio $s/v$ of the partition for a high parallel efficiency of

$$\text{efficiency} = 1/\big(1 + \frac{C_2}{C_1}(\frac{1}{v} t_{\text{startup}} + \frac{s}{v} * t_{\text{bandwidth}})\big) \,. \tag{7}$$

While the lowest continuous surface to volume ratio is obtained for the sphere by $s = \sqrt[d]{2d^{d-1}\frac{\pi^{d/2}}{\Gamma(d/2)}} v^{(d-1)/d}$, we usually deal with partitions aligned with the mesh. Hence the cube with $s = 2dv^{(d-1)/d}$ is of interest. In general we regard estimates of type

$$s \;\leq C_{\text{part}} \cdot v^{(d-1)/d} \tag{8}$$

with low constants $C_{\text{part}}$ as optimal.

## 7 Estimates for Space-Filling Curves

The estimate for the locality of a discrete space-filling curve $F$ we will use with $F : [1, \ldots, k^d] \mapsto [1, \ldots, k]^d$ is of type

$$\|F(x) - F(y)\|_2 \leq C \sqrt[d]{|x - y|} \,. \tag{9}$$

Gotsman and Lindenbaum[37] give an upper bound $C = (d+3)^{d/2} 2^d$ for the Hilbert curve and tighter bounds for $C = 6\frac{2}{3}$ for $d = 2$ and $C = 23$ for $d = 3$, which has been improved[38]. Analogous estimates have been derived for the Hilbert curve[39] and the Peano curve[40]. It turns out that a version of the Sierpinski curve, also called H-index gives even better constants[41,42].

Lemma 1: *Given a connected discrete space-filling curve $F$ on a domain $[1, \ldots, k]^d$ and a partition $F([j, \ldots, j + v - 1])$ of $v$ nodes, the surface $s$ of the partition is bounded by (8). The constant $C_{\text{part}}$ depends on the curve.*
The proof is based on (9) and the connectedness of the partition. It is sufficient to consider $s$ of the bounding box.

This lemma does not hold for curves of Lebesgue type which are also called *bit-interleaving*[43], because the discrete partitions tend to be disconnected. However, we can generalize the situation to unstructured and adaptively refined meshes by the following construction: We create an enumeration of a mesh by some heuristic in order to obtain a 'local' discrete space-filling curve. Then we perform mesh refinement by some geometric refinement rules[23,44]. Each element $E_j$ of the coarse grid is substituted by several smaller elements $E_{j,k}$. The enumeration is changed such that it cycles through these new elements $E_{j,k}$ right after the elements $E_{j-1}$ or $E_{j-1,k}$. This leads in the limit to a continuous space-filling curve[33,30,45]. Alternatively a standard continuous space-filling curve can be super-imposed onto the grid[29,32].

Corollary 2: *Estimate (8) also holds for a space-filling curve partitioning of a (quasi-) uniform mesh by superposition of $f$ or mesh dependent construction of $f$.*

Estimate (7) combined with corollary 2 gives a parallel efficiency for large problems of

$$\text{efficiency} = 1/\big(1 + \frac{C_2 C_{\text{part}} t_{\text{bandwidth}}}{C_1} \cdot \frac{p}{n^{1/d}}\big) \; . \tag{10}$$

This implies optimal parallel efficiency for very large problems, $n \to \infty$. Estimate (10) holds for a code for the solution of partial differential equations in the steps of setting up an equation system, a single matrix multiply and a fixed number of Krylov iterations. Furthermore, using the same space-filling curve on all grid levels, this also holds for an additive multigrid implementation and for standard multigrid if we neglect terms $\log n \cdot t_{\text{startup}}$ proportional to the number of grid levels. For the scalability of a global PDE solver an $O(n)$ multigrid solver is essential. Solvers with higher than linear complexity may scale in $p$ like (10) but scale completely different in $n$. Under suitable conditions, the estimates can be generalized to adaptively refined grids[46].

## 8 Key-Based Addressing

Instead of linked lists or trees, we propose to use *hash storage* techniques[47]. First we describe a *key based* addressing scheme. The entity (a node) is stored in an abstract vector, where it can be retrieved by its key. Furthermore it is possible to decide, whether a given key is stored in the table or not, and it is possible to loop over all keys stored in the vector. In order to reduce the amount of storage of the grid, we omit any pointers and use keys instead. For a (hyper-) cube shaped domain $\Omega = [0,1]^d$, we can use the coordinates of a node for addressing purposes. The coordinates (and the keys) of hierarchical son nodes and father nodes can easily be computed from the node's coordinates. The computation of neighbor nodes requires special care, because it is not immediately clear, where to look for the node. Given a one-irregular grid with hanging nodes, for example, a neighbor node can be located in the distance of $h$ or $2h$ from the node with a local step-size $h$, see Figure 1 (right). In the worst case this results in two vector look-up operations, one in distance $h$ along a coordinate direction and, if it was unsuccessful, one look-up in distance $2h$ [29]. Similar key based addressing schemes can be obtained for other grid refinement procedures and for different domains[33,48].

Key based addressing does simplify the implementation of a sequential, adaptive code. Now, we generalize the concept of key addressing and hash tables to the parallel case.

The idea is to store the data in a hash table located on the local processor. However, we use global keys, so a ghost copy of the node may also reside in the hash table of a neighbor processor. Furthermore we base the code on the space-filling curve partitions of the previous section. The position of a node on the space-filling curve, along with the known partition, defines the home processor of a node. Given a node on a processor, it is easy to determine to which processor the node belongs to. If a node occurs, which does not belong to the processor, it must be a ghost copy, and it is computable where to find its original.

The next idea is to combine the position on the space-filling curve with the hash key[31,34,49]. The computation of the position on the curve can be computed for any given coordinate tuple. It is a unique mapping $[0,1]^d \to [0,1]$ similar to the mapping required for hash keys. The position can be used as a key. Furthermore, for the construction of the hash table, we need a hash function. This can be any mapping $[0,1] \to [0,m]$ with a large integer number $m$, preferably prime. Many cheap functions related to pseudo-random numbers will do here. Modifications of the hash function can improve the cache performance of the code: Space-filling curves introduce locality in the key addressing scheme, which is used for the parallelisation of the code. Exploiting the data locality once again on the local processor, one can optimize the usage of secondary disk storage and of the memory hierarchy of caches, which is difficult otherwise[50].

This framework for the parallelisation of adaptive codes originally has been invented for particle methods[34] and has been generalized to programming environments for some grand challenge PDE projects[31]. Multigrid methods have been considered in Ref. 29, 36.

## 9 Numerical Experiments

| time | | processors | | | | | | |
|---|---|---|---|---|---|---|---|---|
| level | nodes | 1 | 4 | 4 uni proc | 16 | 64 | 256 | 512 |
| 4 | 289 | 0.81 | 1.68 | 1.42 | 2.31 | 4.50 | 10.67 | 11.36 |
| 5 | 1089 | 9.51 | 4.02 | 3.74 | 3.40 | 5.35 | 10.66 | 24.79 |
| 6 | 4225 | 45.17 | 12.97 | 12.48 | 6.90 | 6.76 | 12.45 | 34.35 |
| 7 | 16641 | 267.7 | 53.08 | 51.77 | 18.28 | 14.15 | 18.15 | 23.59 |
| 8 | 66049 | 1722 | 302.4 | 289.3 | 72.92 | 29.20 | 33.10 | 32.58 |
| 9 | 263169 | | 1891 | 1792 | 419.0 | 95.47 | 58.31 | 67.26 |

Table 1. Poisson problem, uniformly refined grids in two dimensions, timing of the multigrid solution on ASCI Blue Pacific.

As test cases for our approach, we consider the Poisson equation and a convection-diffusion problem. We use a finite difference discretisation, where the degrees of freedom are associated with the nodes, and the differential operator is defined on the edges connecting the nodes. In a similar fashion the sparse grid discretisation and the additive multigrid can be defined.

All numbers reported are scaled CPU times measured on a Cray T3E parallel computer with 300MHz Alpha 21164 processors (T3E-600) and on ASCI Blue Pacific (technology refresh, PowerPC 604e, 332 MHz). We use the portable message passing MPI programming interface and its intrinsic timing routines.

| time | | processors | | | | | | |
|---|---|---|---|---|---|---|---|---|
| level | nodes | 1 | 4 | 4 uni proc | 16 | 64 | 256 | 512 |
| 3 | 729 | 7.41 | 5.55 | 4.69 | 4.64 | 8.26 | 17.34 | 26.51 |
| 4 | 4913 | 142.8 | 55.85 | 53.65 | 17.19 | 12.63 | 20.65 | 33.76 |
| 5 | 35937 | 5617 | 1088 | 1036 | 155.1 | 44.01 | 36.18 | 65.32 |
| 6 | 274625 | | | | 3116 | 383.4 | 110.5 | 106.6 |

Table 2. Poisson problem, uniformly refined grids in three dimensions, timing of the multigrid solution on ASCI Blue Pacific.

| time | processors | | | | | |
|---|---|---|---|---|---|---|
| nodes | 1 | 4 | 16 | 64 | 128 | 256 |
| 1089 | 5.08 | 1.27 | 0.72 | 0.64 | 0.84 | 1.30 |
| 1662 | 5.85 | 2.01 | 0.97 | 0.72 | 0.86 | 1.33 |
| 2745 | 10.7 | 3.26 | 1.37 | 0.85 | 0.94 | 1.38 |
| 4834 | 20.3 | 5.84 | 2.01 | 1.08 | 1.08 | 1.46 |
| 8915 | 39.8 | 10.9 | 3.38 | 1.42 | 1.26 | 1.56 |
| 16948 | 78.5 | 39.7 | 5.68 | 2.08 | 1.66 | 1.78 |
| 32788 | 157 | 77.7 | 10.7 | 3.34 | 2.30 | 2.14 |
| 64251 | | | 20.7 | 5.97 | 3.62 | 2.80 |
| 126810 | | | | 10.9 | 6.14 | 4.12 |
| 251468 | | | | | 11.2 | 6.64 |
| 500135 | | | | | 21.2 | 11.7 |
| 996531 | | | | | 41.0 | 21.8 |
| 1988043 | | | | | 80.6 | 41.4 |

Table 3. Poisson problem, adaptively refined grids in two dimensions, timing of the multigrid solution on a Cray T3E-600.

| time | processors | | | | | |
|---|---|---|---|---|---|---|
| nodes | 1 | 4 | 16 | 64 | 128 | 256 |
| 35937 | 291 | 85.6 | 29.6 | 11.2 | 7.61 | 5.94 |
| 50904 | 423 | 129 | 41.0 | 14.8 | 10.1 | 7.17 |
| 89076 | 405 | 236 | 71.2 | 24.6 | 14.6 | 9.98 |
| 189581 | | | 154 | 49.7 | 29.2 | 17.2 |
| 460421 | | | | 109 | 61.1 | 35.6 |
| 1201650 | | | | | 142 | 77.2 |
| 3251102 | | | | | 345 | 188 |

Table 4. Poisson problem, adaptively refined grids in three dimensions, timing of the multigrid solution on a Cray T3E-600.

| nodes | 1 | 4 | 4 uni proc | 16 | 64 | 256 | 512 |
|---|---|---|---|---|---|---|---|
| 59049 | 210.6 | 89.07 | 85.97 | 33.91 | 12.13 | 20.95 | 132.3 |
| 452709 | 2989 | 1462 | 1385 | 556.6 | 198.1 | 63.13 | 37.77 |
| 2421009 | | | | | 2290 | 768.5 | 458.9 |

Table 5. Poisson problem, uniformly refined sparse grids in ten dimensions, timing of the iterative solver on ASCI Blue Pacific.

| level | time nodes | 1/h | processors 1 | 4 | 16 | 32 | 64 | 128 | 256 | 512 |
|---|---|---|---|---|---|---|---|---|---|---|
| 2 | 81 | 4 | 0.04 | 0.06 | 0.10 | 0.14 | 0.16 | | | |
| 3 | 225 | 8 | 0.25 | 0.20 | 0.38 | 0.54 | 0.57 | 0.59 | | |
| 4 | 593 | 16 | 1.31 | 0.71 | 0.89 | 1.24 | 1.53 | 1.87 | 2.34 | 3.26 |
| 5 | 1505 | 32 | 10.0 | 3.66 | 3.03 | 3.67 | 5.75 | 5.28 | 7.17 | 9.97 |
| 6 | 3713 | 64 | 52.5 | 20.1 | 12.8 | 11.9 | 13.1 | 15.4 | 20.5 | 29.9 |
| 7 | 8961 | 128 | 158 | 58.2 | 29.6 | 22.6 | 20.9 | 23.0 | 27.9 | 40.6 |
| 8 | 21249 | 256 | 473 | 192 | 66.6 | 44.9 | 67.6 | 32.2 | 36.7 | 94.0 |
| 9 | 49665 | 512 | 1426 | 396 | 156 | 95.4 | 62.0 | 48.6 | 48.2 | 102.2 |
| 10 | 114689 | 1024 | 4112 | | | | 125 | 85.9 | 68.5 | |

Table 6. Convection-diffusion problem, uniformly refined sparse grids in three dimensions, timing of the iterative solution on a Cray T3E-600.

| level | time nodes | 1/h | processors 1 | 4 | 4 uni proc | 16 | 64 | 256 | 512 |
|---|---|---|---|---|---|---|---|---|---|
| 4 | 593 | 16 | 0.67 | 0.47 | 0.42 | 0.46 | 0.54 | 0.80 | 0.66 |
| 5 | 1505 | 32 | 3.97 | 2.15 | 2.01 | 1.43 | 1.57 | 8.56 | 2.64 |
| 6 | 3713 | 64 | 23.33 | 11.14 | 10.81 | 5.70 | 4.80 | 19.63 | 9.34 |
| 7 | 8961 | 128 | 71.92 | 34.06 | 32.20 | 11.60 | 7.47 | 9.00 | 11.60 |
| 8 | 21249 | 256 | 208.8 | 90.62 | 86.40 | 27.33 | 12.41 | 10.89 | 13.75 |
| 9 | 49665 | 512 | 619.0 | 245.8 | 233.6 | 70.94 | 27.76 | 24.84 | 47.10 |
| 10 | 114689 | 1024 | 1845 | 740.5 | 693.0 | 189.2 | 66.36 | 45.92 | 62.85 |
| 11 | 262145 | 2048 | | | 2184 | 568.2 | 149.2 | 99.44 | 86.30 |
| 12 | 593921 | 4096 | | | | 1766 | 343.2 | 143.0 | 150.6 |
| 13 | 1335297 | 8192 | | | | | 888.1 | 275.0 | 222.3 |

Table 7. Convection-diffusion problem, uniformly refined sparse grids in three dimensions, timing of the iterative solution on ASCI Blue Pacific.

Tables 1 and 2 show execution times and scaling for uniform grids in two and three dimensions with a multigrid solver. These times give the wall clock times for the solution of the equation system, on different levels of grids and on different numbers of processors. We assume a constant number of iterations within a nested iteration. We consider also adaptively refined grids of the unit square and the unit cube. The grids are refined toward the corner $\vec{0}$. Tables 3 and 4 depict times in the adaptive grid refinement case of the Poisson equation in two and three dimensions.

$$-\Delta u + \vec{v} \cdot \nabla u = f \tag{11}$$

A convection-diffusion problem (11) on an uniform, ten-dimensional sparse grid can be found in Table 5. Adaptively refined three dimensional sparse grid solvers are shown in Table 6 and 7, where both hardware platforms can be compared.

We obtain a scaling of about a factor 2 from one level to the next finer level, which means 2 times more unknowns on the next finer level. Increasing the number of processors speeds up the computation accordingly. Again we observe scalability of the algorithm. In order to use all processor efficiently the grid has to be fine enough, i.e. it has to possess a large number of unknowns.

## 10 Conclusion

In this paper we gave a survey of the basic ingredients of an efficient solver for self-adjoint elliptic PDEs, i.e. multilevel solvers, adaptive grid refinement and parallelisation. We focused on the interplay between these ingredients and tried to illustrate how they can be glued together into an adaptive parallel multilevel method. Here we proposed the application of hash storage techniques for data management and the use of space-filling curves for load balancing in the parallel version of the algorithms and presented a version of a parallel adaptive multilevel method based on these approaches.

Note finally that load balancing for adaptive multilevel solvers with space filling curves can be obtained (after slight modifications) in an analogous way and with analogous results also for the case of general unstructured grids. This is actual work in progress.

## Acknowledgments

The computing resources were generously provided by the John von Neumann Institute for Computing (NIC) at Research Centre Jülich (Cray T3E-600) and the Institute for Scientific Computing Research (ISCR) of the Lawrence Livermore National Laboratory (ASCI Blue Pacific).

## References

1. J. H. Bramble, J. E. Pasciak, and J. Xu. Parallel multilevel preconditioners. *Math. Comp.*, 55(191):1–22, 1990.
2. P. Leinen. *Ein schneller adaptiver Löser für elliptische Randwertprobleme auf Seriell- und Parallelrechnern*. Doktorarbeit, Universität Dortmund, 1990.
3. C. E. Grosch. Poisson solvers on large array computers. In B. L. Buzbee and J. F. Morrison, editors, *Proc. 1978 LANL Workshop on Vector and Parallel Processors*, 1978.
4. A. Brandt. Multigrid solvers on parallel computers. In M. H. Schultz, editor, *Elliptic Problem Solvers, Proc. Conf., Santa Fe*, pages 39–83. Academic Press, San Diego, CA, 1981.
5. O. A. McBryan, P. O. Frederickson, J. Linden, A. Schüller, K. Solchenbach, K. Stüben, C.-A. Thole, and U. Trottenberg. Multigrid methods on parallel computers – a survey of recent developments. *IMPACT Comput. Sci. Engrg.*, 3(1):1–75, 1991.
6. J. Bey. Analyse und Simulation eines Konjugierte-Gradienten-Verfahrens mit einem Multilevel Präkonditionierer zur Lösung dreidimensionaler elliptischer Randwertprobleme für massiv parallele Rechner. Diplomarbeit, RWTH Aachen, 1991.
7. G. Zumbusch. Adaptive parallele Multilevel-Methoden zur Lösung elliptischer Randwertprobleme. SFB-Report 342/19/91A, TUM-I9127, SFB 342, TU München, Munich, 1991.
8. S. A. Smolyak. Quadrature and interpolation formulas for tensor products of certain classes of functions. *Dokl. Akad. Nauk SSSR*, 148:1042–1045, 1963.
9. V. N. Temlyakov. Approximation of functions with bounded mixed derivative. *Proc. of the Steklov Institute of Mathematics*, 178:121, 1989.

10. J. A. Cavendish, W. J. Gordon, and C. A. Hall. Ritz-Galerkin approximations in blending function spaces. *Numer. Math.*, 26:155–178, 1976.
11. Din' Zung. Number of integral points in a certain set and the approximation of functions of several variables. *Math. Notes*, 36:736–744, 1984.
12. C. B. Liem, T. Lu, and T. M. Shih. *The Splitting Extrapolation Method: A New Technique in Numerical Solution of Multidimensional Problems*, volume 7 of *Applied Mathematics*. World Scientific, Singapore, 1995.
13. P. W. Hemker. Sparse-grid finite-volume multigrid for 3D-problems. *Adv. Comput. Math.*, 4(1–2):83–110, 1995.
14. C. Zenger. Sparse grids. In W. Hackbusch, editor, *Parallel Algorithms for Partial Differential Equations, Proc. 6th GAMM- Semin., Kiel*, number 31 in Notes on Numerical Fluid Mechanics, pages 241–251. Vieweg, Braunschweig, 1991.
15. M. Griebel, M. Schneider, and C. Zenger. A combination technique for the solution of sparse grid problems. In R. Beauwens and P. De Groen, editors, *Iterative Methods in Linear Algebra. Proc. of the IMACS int. symposium, Brussels*, pages 263–281. North-Holland, Amsterdam, 1992.
16. M. Griebel. Adaptive sparse grid multilevel methods for elliptic PDEs based on finite differences. *Computing*, 61(2):151–179, 1998.
17. T. Schiekofer. *Die Methode der Finiten Differenzen auf dünnen Gittern zur Lösung elliptischer und parabolischer partieller Differentialgleichungen*. Doktorarbeit, Universität Bonn, Inst. für Angew. Math., 1998.
18. M. Griebel. The combination technique for the sparse grid solution of PDEs on multiprocessor machines. *Parallel Processing Letters*, 2:61–70, 1992.
19. G. Faber. Über stetige Funktionen. *Mathematische Annalen*, 66:81–94, 1909.
20. A. Harten. Multi-resolution representation of data: A general framework. *SIAM J. Numer. Anal.*, 33:1205–1256, 1995.
21. M. Griebel and G. Zumbusch. Adaptive sparse grids for hyperbolic conservation laws. In M. Fey and Rolf Jeltsch, editors, *Hyperbolic Problems: Theory, Numerics, Applications. Proc. of the 7th int. conf., Zürich*, volume 1 of *ISNM 129*, pages 411–422. Birkhäuser, Basel, 1999. Proc. 7th Int. Conf., Zürich.
22. A. Pothen. Graph partitioning algorithms with applications to scientific computing. In D. E. Keyes, A. Sameh, and V. Venkatakrishnan, editors, *Parallel Numerical Algorithms. Proc. of the workshop, Hampton*, pages 323–368. Kluwer, Dordrecht, 1997.
23. P. Bastian. *Parallele Adaptive Mehrgitterverfahren*. Skripten zur Numerik. Teubner, Stuttgart, 1996.
24. J. Dubinski. A parallel tree code. *New Astronomy*, 1:133–147, 1996.
25. G. Karypis and V. Kumar. Multilevel k-way graph partitioning for irregular graphs. *J. Parallel Distributed Comput.*, 48(1):96–129, 1998.
26. J. De Keyser and D. Roose. Partitioning and mapping adaptive multigrid hierarchies on distributed memory computers. Technical Report TW 166, Univ. Leuven, Dept. Computer Science, 1992.
27. A. Pothen, H. D. Simon, and K.-P. Liou. Partitioning sparse matrices with eigenvectors of graphs. *SIAM J. Matrix Anal. Appl.*, 11(3):430–452, 1990.
28. G. L. Miller, S.-H. Teng, W. Thurston, and S. A. Vavasis. Geometric separators for finite-element meshes. *SIAM J. Sci. Comput.*, 19(2):364–386, 1998.

29. M. Griebel and G. Zumbusch. Hash-storage techniques for adaptive multilevel solvers and their domain decomposition parallelization. In J. Mandel, C. Farhat, and X.-C. Cai, editors, *Domain decomposition methods 10. The 10th int. conf., Boulder*, volume 218 of *Contemp. Math.*, pages 271–278. AMS, Providence, Rhode Island, 1998.
30. J. T. Oden, A. Patra, and Y. Feng. Domain decomposition for adaptive $hp$ finite element methods. In D. E. Keyes and J. Xu, editors, *Domain decomposition methods in scientific and engineering computing. Proc. of the 7th int. conf. on domain decomposition, Pennsylvania State University*, volume 180 of *Contemp. Math.*, pages 295–301. AMS, Providence, Rhode Island, 1994.
31. M. Parashar and J. C. Browne. On partitioning dynamic adaptive grid hierarchies. In *Proc. of the 29th Annual Hawai Int. Conf. on System Sciences*, pages 604–613, 1996.
32. J. R. Pilkington and S. B. Baden. Partitioning with spacefilling curves. Technical Report CS94-349, UCSD, Dept. Computer Science, 1994.
33. S. Roberts, S. Kalyanasundaram, M. Cardew-Hall, and W. Clarke. A key based parallel adaptive refinement technique for finite element methods. In B. J. Noye, M. D. Teubner, and A. W. Gill, editors, *Proc. Computational Techniques and Applications: CTAC '97*, pages 577–584. World Scientific, Singapore, 1998.
34. M. S. Warren and J. K. Salmon. A portable parallel particle program. *Comput. Phys. Commun.*, 87(1–2):266–290, 1995.
35. H. Sagan. *Space-Filling Curves*. Springer, New York, 1994.
36. M. Griebel and G. Zumbusch. Parallel adaptive subspace correction schemes with applications to elasticity. *Comput. Methods Appl. Mech. Engrg.*, 184(2–4):303–332, 2000.
37. C. Gotsman and M. Lindenbaum. On the metric properties of discrete space-filing curves. *IEEE Trans. Image Processing*, 5(5):794–797, 1996.
38. J. Alber and R. Niedermeier. On multidimensional curves with Hilbert property. *Theory Comput. Systems*, 33(4):295–312, 2000.
39. Q. F. Stout. Topological matching. In *Proc. 15th ACM Symp. on Theory of Computing*, pages 24–31, 1983.
40. A. M. Garsia. Combinatorial inequalities and smoothness of functions. *Bull. Am. Math. Soc.*, 82:157–170, 1976.
41. G. Chochia and M. Cole. Recursive 3D mesh indexing with improved locality. In *Proc. HPCN '97*, number 1225 in Lecture Notes in Computer Science, pages 1014-1015. Springer, Berlin, Heidelberg, 1997.
42. R. Niedermeier, K. Reinhardt, and P. Sanders. Towards optimal locality in mesh-indexings. In *Proc. of the 11th Int. Symposium on Fundamentals of Computation Theory*, number 1279 in Lecture Notes in Computer Science, pages 364–375. Springer, Berlin, Heidelberg, 1997.
43. I. Beichl and F. Sullivan. Interleave in peace, or interleave in pieces. *IEEE Computational Science & Engineering*, 5(2):92–96, 1998.
44. J. Bey. Simplicial grid refinement: On Freudenthal's algorithm and the optimal number of congruence classes. *Numer. Math.*, 85(1), 2000.
45. G. Heber, G. R. Gao, and R. Biswas. Self-avoiding walks over adaptive unstructured grids. *Concurrency - Practice and Experience*, 12:85–109, 2000.
46. G. Zumbusch. Load balancing for adaptively refined grids. *PAMM, Proc. Appl.*

*Math. Mech.*, 1, 2001. in press.
47. D. E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley, Reading, MA, 1975.
48. L. Stals. *Parallel Implementation of Multigrid Methods*. PhD thesis, Australian National Univ., Dept. of Mathematics, 1995.
49. J.. K. Salmon, M. S. Warren, and G. S. Winckelmans. Fast parallel tree codes for gravitational and fluid dynamical N-body problems. *Int. J. Supercomputer Appl.*, 8(2):129–142, 1994.
50. C. C. Douglas. Caching in with multigrid algorithms: problems in two dimensions. *Paral. Alg. Appl.*, 9:195–204, 1996.